Python Tutorial

Release 3.7.0

Creator

Guido van Rossum

LEARNINGVERSE

-Happy Learning

Introduction

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

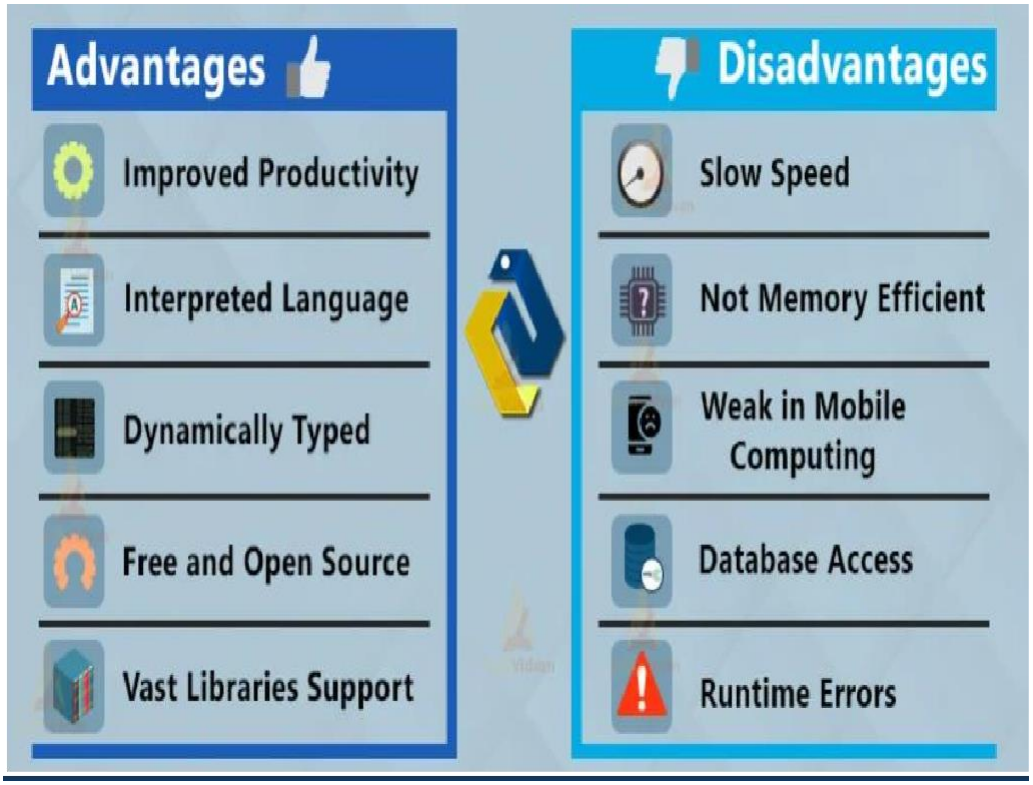
- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes.

Other programming languages often use curly-brackets for this purpose.

Advantages and Disadvantages of Python Programming Language



Python is a **simple** and **versatile Programming language**. It is a great choice for beginners up to professionals. Although it has some disadvantages, we can observe that the advantages exceed the disadvantages. Even Google has made Python one of its primary programming languages.

Syntax

- **Comments are marked by #**
  For Example:
  x += 2 #shorthand for x = x+2
  Note: Python does not have any syntax for multi-line comments, such as the /*…….*/ syntax used in C and C++, though multi-line strings are often used as a replacement for multi-line comments
- **End of the line terminates a statement**
  For Example:
  num = 5
  This is an assignment operation, where we've created a variable named **num** and assigned it the value **5**. Notice that the end of this statement is simply marked by the end of the line.
  In Python, if you'd like a statement to continue to the next line, it is possible to use the "\" marker to indicate this:
  For Example:

```
x = 1+2+3+4+\
    5+6+7+8
```

- **Indentation: Whitespace Matters!**

```
for i in range(10):
        if i%2==0:
                print(“even”)
        else:
                print(“odd”)
```

  This is a compound control-flow statement including a loop and a conditional
- **Whitespace within the line doesn’t matter**
  For example, all three of these expressions are equivalent:
  x=x+2
  x = x + 2
  x = x + 2

Variables

Python Variable is containers which store values. Python is not "statically typed". We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it. A Python variable is a name given to a memory location. It is the basic unit of storage in a program.

Ex:

```
var = "learningverse"
print(var)
```

OUTPUT

learningverse

Rules for creating variable in python

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- Variable names are case-sensitive (name, Name and NAME are three different variables).
- The reserved words(keywords) cannot be used naming the variable.

Re-declare the Variable:

```
var = "learningverse"
print("var : "+var)
var = "learning website"
print("var after re-declaring : "+var)
```

OUTPUT

var : learningverse

var after re-declaring : learning website

Assigning value to multiple variables

```
a=b=10
print(a)
print(b)
```

10

10

How does + operator work with variables?

```
a = 10
b = 20
print(a+b)
a = “learning”
b = “verse”
print(a+b)
```

OUTPUT

```
30
learningverse
```

Global and Local Python Variables

Local variables are the ones that are defined and declared inside a function. We cannot call this variable outside the function.

Ex:

```
def fun():
    s = “welcome to learningverse”
    print(s)
fun()
```

OUTPUT

```
welcome to learningverse
```

Global variables are the ones that are defined and declared outside a function, and we need to use them inside a function.

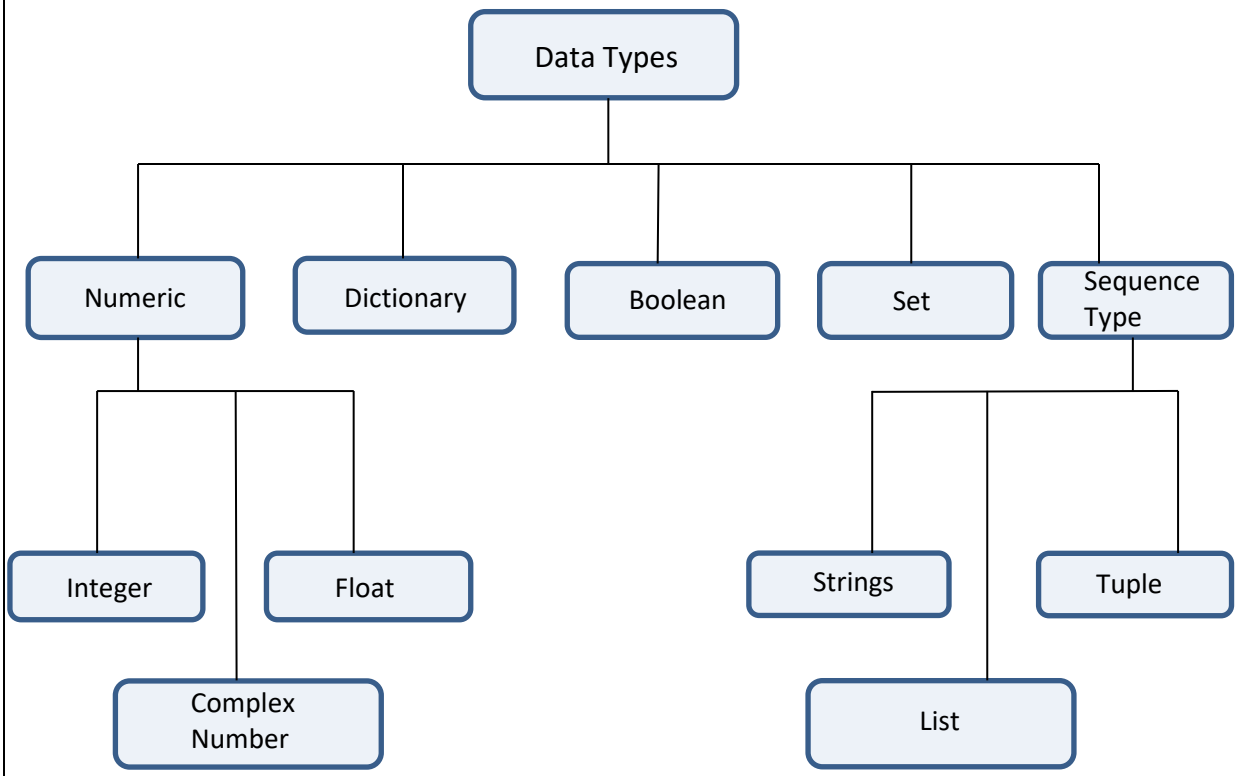
Ex:

```
def fun():
    print(s)
s = “welcome to learningverse”
fun()
```

OUTPUT

```
welcome to learningverse
```

Data types



Numeric

- **Int**
  Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.
- **Float**
  Float, or "floating point number" is a number, positive or negative, containing one or more decimals.
- **Complex**
  Complex numbers are written with a "j" as the imaginary part

Ex:

```
x = 1
y = 2.8
z = 1j
print(type(x))
print(type(y))
print(type(z))
```

OUTPUT
<class ‘int’>
<class ‘float’>
<class ‘complex’>

Sequence Type

- ### Strings

  Strings in python are surrounded by either single quotation marks, or double quotation marks. 'hello' is the same as "hello".

  ```
  a= "Hello"
  print(a)
  OUTPUT
  Hello
  ```

- ### List

  Lists are used to store multiple items in a single variable.
  Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

  ```
  thislist=["apple", "banana", "cherry"]
  print(thislist)
  OUTPUT
  [‘apple’, ‘banana’, ‘cherry’]
  ```

- ### Tuple

  Tuples are used to store multiple items in a single variable.A tuple is a collection which is ordered and **unchangeable**. Tuples are written with round brackets.
  Ex:
  thistuple = ("apple", "banana", "cherry")
  print(thistuple)
  OUTPUT
  (‘apple’, ‘banana’, ‘cherry’)

  #### Tuple Items

  Tuple items are ordered, unchangeable, and allow duplicate values.
  Tuple items are indexed, the first item has index [0], the second item has index [1] etc

  #### Allow Duplicates

  Since tuples are indexed, they can have items with the same value

Dictionary

Dictionaries are used to store data values in key:value pairs.
A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
For Ex:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
print(thisdict[“brand”])
```

OUTPUT
{‘brand’: ‘Ford’, ‘model’: ‘Mustang’, ‘year’: 1964}
Ford

Boolean Values

Booleans represent one of two values: True or False.
In programming you often need to know if an expression is True or False.
You can evaluate any expression in Python, and get one of two answers, True or False.
When you compare two values, the expression is evaluated and Python returns the Boolean answer.
For Ex:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

OUTPUT
True
False
False

SET

Sets are used to store multiple items in a single variable.
Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.
A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
For Ex:

```
thisset = {“apple”, “banana”, “cherry”, “apple”}
print(thisset)
```

OUTPUT
{‘apple’, ‘banana’, ‘cherry’}

Python Conditional Statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a>=b

Example:

If

```
a = 33
b = 200
if b > a:
        print("b is greater than a")
```

Elif

The elif keyword is pythons way of saying "if the previousconditions were not true, then try this condition".

Else

The else keyword catches anything which isn't caught by the preceding conditions.

Example:

```
a = 200
b=3
if b>a:
        print(“b is greater than a”)
elif b==a:
        print(“b is equal to a”)
else:
        print(“b is less than a”)
```

OUTPUT

b is less than a

Python Loops

Python has two primitive loop commands:

- while loops
- for loops

The while Loop

With the while loop we can execute a set of statements as longas a condition is true.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Break Statement

With the break statement we can stop the loop even if the while condition is true:

Example:

```
i = 1
while i < 6:
   print(i)
   if (i == 3):
        break
  i += 1
```

For Loops

A for loop is used for iterating over a sequence(that is either a list, a tuple, a dictionary, a set, or a string).

Example:

```
fruits = [“apple”, “banana”, “cherry”]
for x in fruits:
if x== “banana”:
        break
print(x)
```

Python Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function.
A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

```
def my_function():
  print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

```
def my_function():
        print(“Hello from a function”)
my_function()
```

OUTPUT
Hello from a function

Arguments

Information can be passed into functions as arguments.
Arguments are specified after the function name, inside the parentheses.
You can add as many arguments as you want, just separate them with a comma.
The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

```
def my_func(fname):
        print(fname)
my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

OUTPUT
Emil
Tobias
Linus

Default Parameter Value

If we call the function without argument, it uses the default value

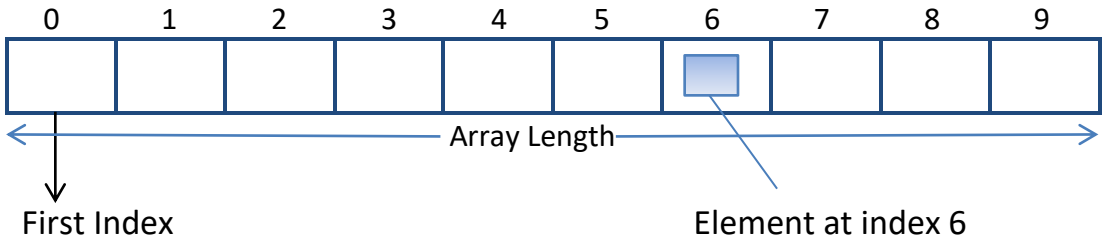
```
def my_function(country= “Norway”):
        print(“I am from” + country)
my_function(“India”)
my_function()
my_function(“Brazil”)
```

OUTPUT
I am from India
I am from Norway
I am from Brazil

Arrays

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

NOTE: Python does not have built-in support for Arrays, but Python Lists can be used instead.



Ex:

```
cars = [“Ford”, “Volvo”, “BMW”]
print(len(cars))
cars.append("Honda")
print(cars)
print(len(cars))
```

OUTPUT

```
3
[“Ford”, “Volvo”, “BMW”, “Honda”]
4
```

Access the element of Array

```
Print(cars[1])
```

Volvo

Length of Array

```
Print(len(cars))
```

3

Access each element in array

```
for i in cars:
        print(cars[i])
```

OUTPUT

Ford

Volvo

BMW

Python Classes/Objects

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword class

```
class MyClass:
 x = 5
print(MyClass)
```

OUTPUT

```
<class ‘ main .MyClass’>
```

Create Object

Now we can use the class named MyClass to create objects:

Ex:

```
class MyClass:
x = 5
p1 = MyClass()
print(p1.x)
```

OUTPUT

```
5
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Ex:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

OUTPUT

```
John
36
```

Note: The __init__() function is called automatically evry time the class is being used to create a new object.

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.
Let us create a method in the Person class:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

OUTPUT

```
Hello my name is John
```

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class. It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class

Ex:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person(“John”, 36)
p1.myfunc()
```

OUTPUT

```
Hello my name is John
```

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.
Parent class is the class being inherited from, also called base class.
Child class is the class that inherits from another class, also called derived class.

Create a Parent Class

Any class can be a parent class, so the syntax is the same as creating any other class:

```
class Person:
  def __init__(self, fname, lname):
     self.firstname=fname
     self.lastname=lname
  def printname(self):
     print(self.firstname, self.lastname)
x=Person("John", "Doe")
x.printname()
```

OUTPUT
John Doe

Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class

```
class Student(Person):
  pass
```

Note: Use the pass keyword when you do not want to add any other properties or methods to the class

Use the super() Function

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
x = Student("Mike", "Olsen")
x.printname()
```